A SOFTWARE RELIABILITY ESTIMATION MODEL

by

Y. R. MURALIDHARA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

February, 1993

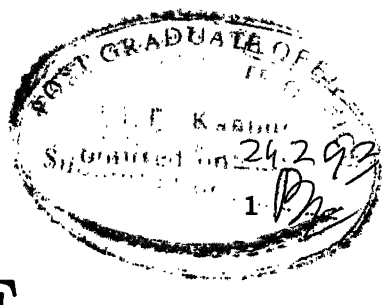
A SOFTWARE RELIABILITY ESTIMATION MODEL

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by
MURALIDHARA Y R

to the
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY
KANPUR
February, 1993

CSE-1993 — M-MUR-SOF



CERTIFICATE

This is to certify that the work contained in the thesis titled **A SOFTWARE RELIABILITY ESTIMATION MODEL** by **MURALIDHARA. Y.R.**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Pankaj Jalote.

Dr. Pankaj Jalote
Associate Professor,
Department of Computer
Science and Engineering
IIT, Kanpur

Acknowledgements

I would like to express my deep gratitude to Dr. Pankaj Jalote, for the constant guidance I obtained from him. The discussions I held with him helped me in understanding the subject better. The moral support he provided during the course of the thesis is unforgettable.

Many persons helped me in solving the problems I encountered in my thesis. The discussion I held with Harish, Deepak Murthy, Rashi and others helped me a lot.

I thank all my teachers, classmates, friends and members of Kannada Sangha for making my stay in I.I.T, a memorable one.

Muralidhara Y R
Kanpur
March 03, 1993.

Abstract

As in any other engineering activity, the development of software systems requires management of resources and maintainance of quality of the product. Quantifiable (measurable) qualities provide an opportunity for efficient management. Many attempts have been made to quantify some of the qualities of the softwares. The quality most extensively considered is reliability. In this thesis some of the important software reliability models are discussed. A software aid is necessary to carry out the complex calculations involved in predicting the reliability quantities using these models. The design and implementation issues involved in the developement of the software aid is discussed in this thesis. A model which uses coverage data to estimate reliability of a software is presented.

Contents

Chapter 1

Software Reliability Modelling

As in any other engineering activity, the development of software systems requires management of resources and maintainance of quality of the product. Quantifiable (measurable) qualities provide an opportunity for efficient management. But unfortunately most of the software quality attributes such as reliability, efficiency, safety, portability, user friendliness, maintainability, readability and others are rather subjective in nature. There have been many attempts to quantify some of the above attributes, of which software reliability is the most extensively considered. These attempts lead to the development of software reliability field.

Software reliability theory has in fact taken a lot from already established field of hardware reliability. Even though there are many inherent difference between software and hardware reliability theories, classical concepts, techniques of hardware reliability theory can be used in the field of software reliability. But a good understanding of software development process is necessary for one to apply the techniques borrowed from the hardware reliability theory.

1.1 Some Definitions

Software is essentially a mapping of some discrete set of inputs to discrete set of outputs. Since, to a large extent software is produced by humans, often there exists a discrepancy between what software can do and what user or the environment wants it to do.

The defects in the software which cause unexpected departure of software from the program requirements are called *software faults.* This departure itself is called *software failure.* The action or omission of the programmer which led to the introduction of fault is called an *error.* The term *software reliability* is defined [AGL85] as the probability of successful execution of the program in the given period of time under given environment.

Two approaches are generally used for detecting the existence of faults namely program proving and program testing. The program testing is the most widely used. Even though testing does not guarantee program correctness, it provides useful information about programs' actual behaviour in its intended environment. Testing provides data about failures which is used by software reliability models.

This chapter gives a brief overview of the theory of software reliability, software reliability models, metrics for the comparison of models and recalibration techniques to refine existing models.

1.2 Software Reliability Growth Problem

The basic problem of software reliability has been explained [BLW88] in terms of time between failures as follows. Times between failures is measured either as the execution time (i.e. CPU time that has been used by the executing program) or as the calendar time between failures. Data about the execution

time between failures $t_1, t_2, \ldots, t_{i-1}$ is provided by the testing. This data is regarded as the realization of random variables $T_1, T_2, \ldots, T_{i-1}$. Using this data expected time for future failures i.e. $T_i, T_{i+1}, \ldots$, are to be predicted.

Software reliability models form only a part of this prediction process. Actually the prediction system can be assumed to comprise of

1. Probability model which specifies the distribution of any subset of the T_js conditional on a (unknown) parameter vector Θ.
2. A statistical inference procedure for Θ involving the use of available data (realization of T_js).
3. A prediction procedure combining 1 and 2 to make probability statements about future T_js.

1.3 Introduction to Reliability Theory

Modeling the software failure process starts with a statistical description of the failure process.

1.3.1 Times Between Failures Description

In times between failures description, the failure process is characterized by the random variable T_i representing the time between(i-1) th and i th failures ; $1 < i <= n$ for some n. Then the reliability function for i th time interval R_i(t) is given by

$$R_i(t) = \text{Prob}[T_i > t] \tag{1.1}$$

i.e , R_i(t) gives the probability that next failure does not occur before time t. Let random variable T_i have a cumulative distribution function (c.d.f) F_i(t) and

probability density f_i(t).

$$F_i(t) = \text{prob}[T_i <= t] = \int_0^t f_i(x).dx \tag{1.2}$$

$$R_i(t) = 1 - F_i(t) \tag{1.3}$$

Some of the distribution functions which are frequently used in reliability theory are, exponential, duane, pareto and weibull.

A quantity of significance, in software reliability is Mean-Time-To-Failure (MTTF). It is defined as the expectation of the random variable T_i. MTTF represents the average time elapsed before next failure occurs in the system.

$$\text{MTTF} = \text{E[T]} = \int_0^\infty t f(t)dt \tag{1.4}$$

It can be shown that MTTF can be computed as

$$\text{MTTF} = \int_0^\infty R(t).dt \tag{1.5}$$

Another significant quantity is median time of failure. It is defined as the value $\bar{t}$ such that

$$F(\bar{t}) = R(\bar{t}) = \frac{1}{2} \tag{1.6}$$

Considering median has an advantage in that it is always well defined, whereas there are certain random variables whose distribution function results in values of MTTF which are not finite.

Measure of proneness to failure, as a function of time passed since previous failure, is given by hazard rate z(t). It is the probability that a failure occurs in the infinitesimal interval [t, $t + \delta$ t] given that no failure has occurred before time t. The equation connecting z(t) with other important quantities are

$$z(t) = \frac{f(t)}{R(t)} \tag{1.7}$$

$$R(t) = e^{-\int z(t)dt} \tag{1.8}$$

1.3.2 Failure Counting Description.

An alternative way of representing the failure process is to consider the cumulative number of failures experienced in the systems up to given time t. Let [n(t)] be the random variable representing the number of failures till time t. The failure process can be completely specified by assigning probability distribution to the random variable n(t), for any t.

As n(t) can assume only integer values, corresponding distribution must be discrete. Usually Poisson and binomial distributions are considered.

1.3.3 Statistical Inference Procedure

Most of the models use maximum likelihood method. Let $T_1, T_2, \ldots, T_n$ be independent random variable with p.d.fs $f_1(t_1/\theta), f_2(t_2/\theta), \ldots, f_n(t_n/\theta)$ where θ is the vector of parameters $\theta_1, \theta_2, \ldots, \theta_k$ that needs to be estimated. Then we define the likelihood function

$$L(\Theta) = g(T_1 = t_1, T_2 = t_2, \ldots, T_n = t_n/\Theta) = \Pi_{i=1}^{n} f_i(t_i/\Theta) \tag{1.9}$$

This likelihood function is the joint p.d.f. of the random variables $T_1, T_2, \ldots, T_n$.

The value Θ_p which maximises $L(\Theta)$ is known as Maximum likelihood estimator of Θ. For large samples maximum likelihood estimators are as good as any other estimators. In certain cases it provides the only alternative for establishing confidence intervals.

Least mean square error method is another well known statistical inference method. It involves minimizing the sum of the squares of the errors of the quantity being measured. The least mean square error estimators provides good alter-

native to maximum likelihood estimators when sample size is small or medium. For such sample they may have lesser variability and approach normality faster.

1.4 Assumptions and Limitations

Each model makes some assumptions about the software failure process so that the model becomes mathematically tractable. Goel [AGL85] has given the typical assumptions made by the models along with the limitations (It should be noted that not all assumptions given below are made by each and every model). The assumptions are as follows:

- Times between failures are independent.

  This assumption is used in all *times between failures* models. In general this would be the case if successive test cases are selected randomly. But usually test case selected may depend on the nature of the fault previously detected. So, it is necessary to introduce at least some amount of independence.

- A detected fault is immediately corrected.

  This assumption is at least implicitly satisfied in many cases. The fault detected is either immediately corrected or the test cases are selected in such a way that the present fault is got around.

- No new faults are introduced during fault removal process.

  This is the most restrictive assumption in reliability models. This assumption is to ensure that the modelled failure process is monotonic in pattern. The only way to satisfy this is to ensure that correction process does not introduce any new errors.

- Failure rate decreases with test time.

This assumption implies that the software gets better with testing in statistical sense. As the testing proceeds the faults are detected and removed. Hence this assumption is reasonable enough.

- Failure rate is proportional to the number of remaining faults.

  This assumption implies that given fault has same chance of being detected in given testing interval between failures. If the test cases are selected in such a way that all paths in the program have equal probability of being executed, then this assumption is reasonable. Otherwise, if some faults lay on rarely executed paths then their detection or non-detection does not affect failure rate appreciably.

- Reliability is function of number of remaining faults.

  This assumption is similar to above assumption, but concerned with operational phase. It implies all remaining faults are equally likely to appear during operational phase. As this is not the case always, the reliability value estimated should be interpreted carefully.

- Time is used as a basis for the failure rate.

  Most models use time as the basis for determining changes in failure rate. This usage assumes that testing is proportional to the time. If testing is not proportional to time other measures such as lines of code tested, number of functions tested, number test cases generated etc., can be used.

- Failure rate increases between failures.

  This would be justifiable assumption only if the testing effort increases in the same way as the failure rate, within a given failure interval.

- Testing is representative of operational usage.

This assumption is necessary to project reliability estimations done with data collected during testing, into operational phase. The test cases are usually selected to satisfy this assumption. As test cases include certain paths that may never be executed in operational phase, reliability measures estimated in testing phase may be bit pessimistic.

1.5 Software Reliability Growth Models

Major steps in developing software reliability model started with the efforts of Jelinski and Moranda [JAM72] and Shooman [SHO72]. In their models hazard rate is assumed to be constant during the time interval between failures and proportional to the number of faults remaining. Hazard rate is assumed to change by a constant amount when a fault is fixed. This constant is estimated using maximum likelihood method in Jelinski Moranda (JM) model. In Shooman model the constant is found out with the help of failure correction profile. In a variant of JM model [MOR75] hazard rate decreases in steps that form a geometric progression.

In another early model proposed by Schick and Wolverton [SAW73] the hazard rate is assumed to be proportional to the product of number of faults remaining and time. Hence the size of changes in hazard rate (at fault correction) increases with time.

Musa [JDM75] presented an execution time model of software reliability. All the models presented before this considered calendar time as the measure of time between failures or they just ignored the kind of time to use. The actual processor time utilized in executing program was found out to be the best measure of failure causing stress. Musa also proposed a calendar component which relates execuction time to calendar time.

Littlewood - Verall [LAV73] took a Bayesian approach to software reliability measurement. They assumed hazard rate as a random variable. The concept of the hazard rate as a random variable reflects uncertainity in the effectiveness of the fault correction process. By changing the parameters of the hazard rate distribution different models can be obtained. One such variant was proposed by Keiller and Littlewood [KLM83].

Goel - Okumoto [GAO79] making assumptions similar to those of JM model, described failure detection as nonhomogeneous Poisson process(NHPP) with an exponentially decaying rate function. The cumulative number of failures detected and the distribution of the number of faults remaining are both turned out to be NHPP.

To reflect situation in which early fixes reduces failure intensity more, Musa -Okumoto proposed logarithmic Poisson model [MAO84]. This model is based on nonhomogeneous Poisson process with an intensity function that decreases exponentially with expected failures.

Probability models proposed so far are classified according to the characterization of failure process [GAO79] as follows.

1. Times between failures models.
2. Failure count models.
3. Input domain models.
4. Fault seeding models.

In the sections to be followed four main software reliability models are described. Jelinski Moranda, Littlewood Verall models are classified under times between failures models. The Goel - Okumoto and Musa - Okumoto models belongs to the class failure count models. Nelson model and Mill's seeding model

belongs to input domain models and fault seeding models respectively. The inputs to the models belonging to first two classes usually have following forms.

1. $t_1, t_2, \ldots, t_{i-1}$ where t_j is the time (cpu time or calendar time) elapsed between the detection of the (j-1)th and the jth error.
2. $t_1', t_2', \ldots, t_{i-1}'$ where t_j' is the time (cpu time or calendar time) elapsed before the detection of the jth error. i.e. $t_i' = \Sigma_{j=1}^{i} t_j$
3. $n_1, n_2, \ldots, n_{i-1}$ where n_j is the number of error occurred in the jth time interval.

It is easy to observe that input form (3) can be obtained from (1) and (2). Form (1) and (2) are interchangeable.

1.5.1 Times Between Failures Model

Jelinski-Moranda de-eutrophication (JM) model

The model proposed by Jelinski and Moranda [JAM72] is one of the earliest and the simplest software reliability models. The JM model makes following assumptions about the software failure process.

1. The times between failures are independent random variables, $T_1, T_2, \ldots$, following an exponential distributions.
2. There are finite number of faults at the beginning of the test phase.
3. Failure rate is uniform between successive failures and is proportional to the current error content (number of faults remaining) of the program being tested.
4. Fault detected is immediately and completely fixed.

From assumption (1) it follows that input to the model is to be in the form (1) given before and also the probability density function for random variable T_i is given by

$$f(t/\lambda_i) = \lambda_i e^{-\lambda_i t} \tag{1.10}$$

From assumption (2) and (3) we have failure rate λ_i

$$\lambda_i = \phi(N - i + 1) \tag{1.11}$$

where N is the total number of faults in the software at the beginning of the test, i is the number of faults detected so far and ϕ is the reduction in failure intensity per failure per fault.

The reliability function is given by

$$R_i(t) = e^{-\lambda_i t} \tag{1.12}$$

Hence the estimate of reliability is given by

$$R_i^p(t) = e^{-\lambda_i t} = e^{-(N^p - i + 1)\phi^p t} \tag{1.13}$$

where N^p and ϕ^p are estimated from available data t_1 , t_2 ,..., t_{i-1} using maximum likelihood method.

The current MTTF is given by

$$\text{MTTF} = \frac{1}{\lambda_i} \tag{1.14}$$

The advantage of this model is that it is very simple to use. It is also fairly accurate for many data sets. But it has some drawbacks too. It is easy to see that failure rate decreases by constant amount ϕ each time a fault is fixed. This is

nothing but the assumption that each fault contributes equally to the unreliability of the program. But this assumption is not always true. Another assumption which may not hold is that faults are fully removed. These assumptions may some time lead to inaccurate predictions.

Littlewood-Verall (LV) model

As in JM model this model [LAV73] assumes exponential distribution for the random variable T_i representing the failure interval time. But the failure intensity is no more a decreasing deterministic function. In LV model it is regarded as a stochastically decreasing function with gamma distribution. Considering successive failure rates as random variables implies the fault fixing process is not considered as perfect. It also reflects the fact that faults are of different sizes. Assumptions of this model are as follows.

Assumption 1: The random variable representing failure interval follows exponential distribution. i.e.

$$f(t_i/\Lambda_i = \lambda_i) = \lambda_i e^{-\lambda_i t} \quad \text{for } (t_i > 0) \tag{1.15}$$

Assumption 2:

$\{\Lambda_i\}$ are assumed to be a sequence of independent stochastically decreasing random variables. This reflects the likelihood not the certainity, that a fix will be effective. It is assumed that

$$f(\lambda_i) = \frac{\Psi(i)^{\alpha}\lambda_i^{\alpha-1}e^{-\Psi(i)\lambda_i}}{\Gamma(\alpha)} \tag{1.16}$$

a gamma distribution with parameters $\alpha, \Psi(\mathrm{i})$.

The function Ψ(i) which is under the control of the user, determines the nature of the reliability growth. If, as usually the case, Ψ(i) is an increasing function of i, then it can be shown that Λ_i s forms a stochastically decreasing sequence. In this model Ψ(i) is taken as $\Psi(i,\beta) = \beta_1 + \beta_2$ i.

Quantities of interest:

The unconditional p.d.f of the time interval can be deduced by the assumption (1) and (2). It is given by

$$f_i(t_i) = \frac{\alpha\Psi(i,\beta^p)^\alpha}{[t + \Psi(i,\beta^p)]^{\alpha+1}} \tag{1.17}$$

The current reliability estimate is given by

$$R_i^p(t) = [\frac{\Psi(i,\beta^p)}{t + \Psi(i,\beta^p)}]^\alpha \tag{1.18}$$

Mean Time To Failure does not exist for $\alpha <= 1$. It is given by

$$MTTF = \frac{\Psi(i)}{\alpha - 1} \quad \alpha > 1 \tag{1.19}$$

Predictions are by Maximum likelihood estimation of parameters α, β_1, β_2 and use of plug-in rule.

Problem with this model lies in complexity of the statistical inference procedure involved in determining the parameters . For estimated parameters, MTTF may not be finite. But for these drawbacks LV model has a great edge over many other models. The comparitive studies of Littlewood [ACL86] have clearly established this. Its treatment of software failure process seems to be "nearer" to the actual phenomenon in many cases. Unlike JM and other models it even permits reliability decay observed in many projects.

1.5.2 Fault Counting Models

Goel-Okumoto NHPP (GO) model

Goel-Okumoto [GAO79] modeled software failure process as a nonhomogeneous Poisson process. This model treats initial error contents as a random variable. Time between k-1 th and k th failure depends on the time to k-1 th failure. In these two respects it differs from the JM model.

The GO model makes following assumptions about the software failure process in order to represent it as an NHPP. They are as follows.

1. The cumulative number of failures up to time t is a random variable n(t); n(0) = 0 i.e. no failures observed at t=0.

2. Number of software failures occurring during non overlapped time intervals do not affect each other.

3. Expected number of faults in the software is finite. i.e. E[n(t)]= μ(t) = a as t tends to ∞.

4. Expected number of software failures in the interval [t, t+δt] is proportional to the expected number of undetected errors.

5. If λ(t) (= μ'(t)) is the failure occurrence rate at time t then the possibility of one failure occurring in the interval [t, t+δt] is λ(t).δt + o(δt) where o(δt)/δt tends to zero as δt tends to zero. The probability of two are more failures occurring in this interval is o(δt).

From assumption (5) we can show that the failure process is an NHPP with a mean function μ(t) i.e.

$$Pr\{n(t) = y\} = \frac{[\mu(t)]^y}{y!} e^{-\mu(t)} \quad \text{for y= 0,1,...} \tag{1.20}$$

From assumptions (1), (3) and (4) we can prove that

$$\mu(t) = a(1 - e^{-bt}) \tag{1.21}$$

where a is the expected number of failures in the system and b is the initial failure intensity. Hence, the failure rate can be expressed as,

$$\lambda(t) = abe^{-bt} \tag{1.22}$$

$$\lambda(\mu) = b(a - \mu) \tag{1.23}$$

Input to this model is in the form (1). i.e $t_1, t2, \ldots$ where t_j is the time between j-1 th and j th failures. From this data cumulative error n(t) can be easily calculated.

Let $T_i (i = 1, 2, \ldots)$ be a random variable representing the ith failure time interval. Define $T_i' (i = 1, 2, \ldots)$ as the random variable representing time to the i th failure, i.e.

$$T_i' = \Sigma_{j=1}^{i} T_j = T_{i-1}' + T_i \tag{1.24}$$

From the above definitions it can be seen that the events $\{n(t) >= i\}$ and $\{T_i' <= t\}$ are equivalent.

Reliability: The reliability of T_i (representing ith failure interval) conditional on the last failure time $T_{i-1}' = t_{i-1}'$ can be obtained as follows.

$$R(t/t_{i-1}') = Pr\{T_i > t/T_{i-1}' = t_{i-1}'\} \tag{1.25}$$

where t gives the time elapsed since last failure. Changing time between failure to failure count description (with the help of equivalence of events$\{n(t) >= i\}$ and $\{T_i' <= t\}$) we can get the conditional reliability as

$$R(t/t_{i-1}') = e^{-\{\mu(t+t_{i-1}')-\mu(t_{i-1}')\}} \tag{1.26}$$

Probability density function: Taking negative derivative of the above equation with respect to conditional probability density function is given as

$$f(t/t_{i-1}') = \lambda(t + t_{i-1}').e^{-\{\mu(t+t_{i-1}')-\mu(t_{i-1}')\}} \tag{1.27}$$

Hazard rate: The conditional hazard rate is given as

$$z(t/t_{i-1}') = \frac{f(t/t_{i-1}')}{R(t/t_{i-1}')} = \lambda(t + t_{i-1}') \tag{1.28}$$

Estimation of the parameters a and b is by maximum likelihood method. Let a^p and b^p be the estimations. Then the quantities of interest are evaluated by substitution (plug-in rule).

From the equations for the failure intensity it can be noted that it is linearly decreasing with the number failures occurred. Hence it follows that GO model assumes that each fault in the software contributes equally to the unreliability of the software. It also assumes perfect fixing of the detected faults . Both these assumptions do not always hold good, thus affecting the predictions.

Musa-Okumoto logarithmic execution (MO) model

The model proposed by Musa and Okumoto [MAO84] views failure process as an NHPP like GO model. But Unlike GO model it assumes reduction in failure

rates are greater for the earlier fixes. This is because of the observation that early failures are caused by the faults lying on most frequently executed paths of the program. This model assumes perfect fixing of detected faults.

Assumptions 1 and 5 of the GO model which are required model failure process as an NHPP are used in this model also. But instead of linear dependence of failure intensity on number failures occurred, MO model assumes failure rate to be an exponential function of the expected number of failures.

$$\lambda(t) = \lambda_0 e^{-\theta\mu(t)} \tag{1.29}$$

where λ_0 and θ are initial failure rate and reduction in the normalized failure intensity per failure respectively.

The above assumption is to reflect the fact that initial failures are BIG.

We know that $\lambda(t) = \mu'(t)$. Substituting for $\lambda(t)$ by the equation given above and solving the resultant differential equation we get

$$\mu(t) = \frac{1}{\theta}.ln(\lambda\theta t + 1) \tag{1.30}$$

therefore,

$$\lambda(t) = \frac{\lambda_0}{(\lambda_0\theta t + 1)} \tag{1.31}$$

Input to the model is in the form(1). But the time intervals $t_1, t_2, \ldots$ are execution times. Musa [MAO84a] has established the superiority of execution time over calendar time when it comes to software reliability models. This model works for calendar times also.

Using the formulae given in the previous section for an NHPP process and substituting in them the formulae for $\lambda(t), \mu(t)$ given above, we have

Conditional reliability,

$$R(t/t'_{i-1}) = \{\frac{\lambda_0\theta t'_{i-1}+1}{\lambda_0\theta(t+t'_{i-1})+1}\}^{\frac{1}{\theta}} \tag{1.32}$$

Conditional p.d.f.

$$f(t/t'_{i-1}) = \frac{\lambda_0}{(\lambda_0\theta t+1)}\{\frac{\lambda_0\theta t'_{i-1}+1}{\lambda_0\theta(t+t'_{i-1})+1}\}^{\frac{1}{\theta}} \tag{1.33}$$

Conditional hazard rate,

$$z(t/t'_{i-1}) = \frac{\lambda_0}{(\lambda_0\theta(t+t'_{i-1})+1)} \tag{1.34}$$

Estimation of parameter λ_0, θ is by maximum likelihood method. By substituting the estimated values λ_0^p, θ^p the reliability and other quantities are determined.

Calendar time component: It is the calendar time which makes sense to the software managers. Hence it is necessary to relate execution time to calendar time. For this purpose following assumptions are made.

Assumption 1: The pace of testing at any time is constrained by one of the three limiting resources. Failure identification(testing) personnel(I), Failure correction(original designer) personnel(F) and Computer time (C).

This assumption is based on the observation that during the beginning of testing large faults are detected, followed by a phase when rate of detection slows down. These two phases are followed by the last phase in which it becomes extremely difficult to find the faults. During these three phases debugging, testing teams and computer resource become bottlenecks.

Denoting the execution time by τ and calendar time by t, above assumption can be written as

$$\frac{dt}{d\tau} = max\{\frac{dt_I}{d\tau}, \frac{dt_F}{d\tau}, \frac{dt_C}{d\tau}\} \tag{1.35}$$

where $\frac{dt_k}{d\tau}$ represents the instantaneous calendar time to execution time ratio that result from the effect of resource constraint k (k= I,F,C).

Assumption 2: The rate of resource expenditure with respect to execution time $\frac{d\chi_k}{d\tau}$ can be approximated by

$$\frac{d\chi_k}{d\tau} = \theta_k + \mu_k \frac{d\mu(t)}{d\tau} \quad \text{k= I,F,C} \tag{1.36}$$

where θ_k is the execution time coefficient of resource expenditure and μ_k is the failure coefficient of resource expenditure.

For this logarithmic model we have

$$\frac{d\chi_k}{d\tau} = \theta_k + \mu_k \frac{\lambda_0}{(\lambda_0 \theta \tau + 1)} \tag{1.37}$$

Assumption 3: The quantities of available resources(P_k) are constant for the remainder of the test period and maximum utilization of the resources (ρ_k) is also constant.

Then we have,

$$\frac{dt_k}{d\tau} = \frac{\frac{d\chi_k}{d\tau}}{\rho_k P_k} \tag{1.38}$$

1.5.3 Fault Seeding Models

Mills seeding model: The most popular and basic fault seeding model is Mill's hyper-geometric model. This model requires that a number of known faults to

be randomly seeded in the program to be tested. The program is then tested for some amount of time. The number of original indigenous faults can be estimated from the number of indigenous and seeded faults uncovered during the test, using hyper-geometric distribution. models suggested by Lipow and Basin are modification of this model.

1.5.4 Input Domain Models

Nelson model: This model assumes the prior knowledge of operational profile. Let n sample inputs are taken from input domain set E whose elements are the inputs needed to make a run. The sampling is done according to the probability distribution P_i where the set $\{P_i\}$ is the user probability distribution. If n_e is the number of inputs that caused a failure then the reliability is given by

$$R_i^p = 1 - \frac{n_e}{n} \tag{1.39}$$

One of the disadvantages is that the test set used during verification may not be the representative of the expected operational usage.

Ramamoorthy and Bastani model:

This model calculates the conditional probability that the program is correct given that it is correct for given set of inputs. The basic assumption is that the outcome of each each test case provides some stochastic information about the points that are close to the test point.

P{program is correct for all points in [a, a+V]/ it is correct for testy cases having successive distances $x_j, j = 1, 2, ..., n-1$ }

$$= e^{-\lambda V} \Pi_{j=1}^{n-1} \frac{2}{1 + e^{-\lambda x_j}} \tag{1.40}$$

3	30	113	81	115	9	2	91
112	15	138	50	77	24	108	88
670	120	26	114	325	55	242	68
422	180	10	1146	600	15	36	4
0	8	227	65	176	58	457	300
97	263	452	255	197	193	6	79
816	1351	148	21	233	134	357	193
236	31	369	748	0	232	330	365
1222	543	10	16	529	379	44	129
810	290	300	529	281	160	828	1011
445	296	1755	1064	1783	860	983	707
33	868	724	2323	2930	1461	843	12
261	1800	865	1435	30	143	108	0
3110	1247	943	700	875	245	729	1897
447	386	446	122	990	968	1082	22
75	482	5509	100	10	1071	371	790
6150	3321	1045	648	5485	1160	1864	4116

Table 1.1: Times between failures data of system T1 [read rowwise]

where λ is a parameter which is deduced from some measure of complexity of source code.

1.6 An Example of Reliability Predictions

In this section an example of statistical inference by maximum likelihood method and prediction of reliability quantities for the models presented in this report is worked out. The input is taken as the sequence of execution times between failures, for the system T1 given by Musa [JDM79]. The data is given in Table 1.6. From these 136 times between failures data , time to failure data T_i's required by GO and MO models are obtained.

1.6.1 Estimation of Parameters

From the equation (1.27) for p.m.f of the times to failures($T_i^{'}$)s we have, the likelihood function as

$$f_{T_1^{'},T_2^{'},\ldots,T_n^{'}}(t_1^{'},t_2^{'},\ldots,t_n^{'}) = \Pi_{i=1}^{n} f(t/t_{i-1}^{'}) = \lambda(t+t_{i-1}^{'}).e^{-\{\mu(t+t_{i-1}^{'})-\mu(t_{i-1}^{'})\}} \tag{1.41}$$

Substituting for $\lambda(t+t_{i-1}^{'})$ and simplifying we have,

$$f_{T_1^{'},T_2^{'},\ldots,T_n^{'}}(t_1^{'},t_2^{'},\ldots,t_n^{'}) = e^{-\mu(t_n^{'})}\Pi_{i=1}^{n} a.b.e^{-b.t_i^{'}} \tag{1.42}$$

As the product given above tends to be very small, we deal with the log likelihood function. The maximizing values of parameters are same in both cases. Taking the log of the likelihood function we have,

$$L(a,b/T^{'}) = nlna + nlnb - a(1-e^{-bt_n}) - b\Sigma_{i=1}^{n} t_i^{'} \tag{1.43}$$

Finding a and b which maximizes the above function for the given set of $t_i^{'}$ we get

$$a^p = 142.88 \quad \text{faults.}$$

$$b^p = 0.000034 \quad /\text{sec.}$$

Similarly the likelihood functions for the other models and the values of the maximizing values of parameters, are given for different models. The data used in all cases remains same.

For JM model the estimated parameter

$$N^p = 141.90 \quad \text{faults.}$$

RELIABILITY QUANTITIES	JM	LV	GO	MO
Expected no. of failures	136.0	–	136.3	136.6
No. of faults remaining	5.9	–	6.5	–
No. of faults in the system	141.9	–	142.8	–
Reliability	0.73	0.31	0.74	0.55
Failure rate	0.000241	0.000837	0.000225	0.000452
Mean Time To Failure	4848.25	1177.3	4552.44	2164.39

Table 1.2: Reliability quantities at 90000 s for the data set given in table 1.6.

$$\phi^p = 0.000035 \ \text{/fault/sec.}$$

The parameters of LV model have following values,

$$\alpha = 7.44$$

$$\beta_1 = 84.47$$

$$\beta_2 = 54.73$$

The parameters of the Musa model turns out to be,

$$\lambda_0 = 0.01083 \ \text{faults/sec.}$$

$$\theta = 0.02335 \ \text{/sec.}$$

1.6.2 Reliability Predictions

Once the parameters are estimated, the calculation of the reliability quantities can be easily done by substitution. The important quantities at the time 90000 seconds are calculated for the four models discussed and tabulated.

It should be noted that for GO model instead of MTTF, instantaneous MTTF (inverse of hazard rate) is considered as it is difficult to calculate MTTF. As

LV and MO models allow for infinite faults total number of faults and faults remaining do not have any meaning.

By looking at the results it seems that JM and GO models are close to each other in terms of performance. This is due to the closeness of the assumptions made by the models. Again the reliability, failure rate and MTTF figures suggest LV and MO to be a bit pessimistic.

1.7 Comparing Models

The fact that the number of software reliability models is very large (more than 40), necessitates the development of comparison criteria. It is also known from the experience that no single model predicts satisfactorily under all circumstances. Many attempts have been made towards developing criteria for comparison purposes. Iannio et al. [ILM84] gave certain comparison criteria which represents the approximate consensus among number of researchers in the field. The criteria are

1. Predictive validity
2. Capability
3. Quality of assumptions
4. Applicability
5. Simplicity

Predictive Validity

Predictive validity is the ability of the model to determine future failure behaviour during either test or operational phase from present or past failure be-

haviour in the respective phases. One way of evaluating the predictive validity can be as follows.

The predictive failure random process is described by $\{m(t), t > 0\}$, representing the number of failure experienced by the time t. Such a counting process is characterized by specifying the distribution of m(t), including the mean value function μ(t).

At the end of time t_q let q failures have been observed. The failure data up to time $t_e(<= t_q)$ is used to estimate the parameters of μ(t). Then the number of failures by t_q can be predicted by substituting the estimates of the parameters in the mean value function $\mu^p(t_q)$, which is compared with actually observed q. This is repeated for various values of t_e .

The plot of normalized relative error $\frac{\mu^p(t_q)-q}{q}$ against normalized time $\frac{t_e}{t_q}$ gives an indication of predictive validity. The error will approach zero as t_e approaches t_q . If points are positive(negative), the model tends to overestimate(underestimate). Predictive validity is a function of both the model and the inference procedure. A method to evaluate the competing software reliability predictions is discussed later.

Capability

Capability refers to the ability of the model to estimate with satisfactory accuracy quantities needed by software managers, engineers and users in planning and managing software development projects or controlling change in operational software systems. The degree of capability is decided by the number of quantities and their relative weightage. The quantities, in approximate order of their importance are

1. Present reliability, MTTF, or failure intensity.

2. Expected date of reaching a specified reliability, MTTF or failure intensity goal.

3. Resources and cost requirements related to the achievement of the foregoing goals.

If the model posses capability for prediction of software reliability in the system design and early development phases, it becomes extremely valuable as it helps in systems engineering and planning purposes.

Quality of Assumption

If it is possible to test the assumption, the degree to which it is supported by actual data is to be considered. If it is not possible to test the assumption then its plausibility from the view point of logical consistency and software engineering experience should be evaluated. Finally the clarity and explicitness of an assumption should be judged to determine whether a model applies to particular circumstances.

Applicability

A model should be judged on its degree of applicability across different software products (size, structure, function etc), different development environments, different operational environments etc. But if a model is excellent for a narrow range of products it should not be eliminated. Models' applicability depends on how they deal with following situations.

- Phased integration of a program (testing before integration of entire program results in failure data associated with partial program).
- Design and requirements changes to the program.
- Classification of severity of failures.

- Ability to handle incomplete failure data or data with measurement uncertainities.
- Operation of same program on computers of different performances.

Simplicity

Models should be simple in following respects.

- Simple and inexpensive procedure to collect data.
- Simple to understand.
- Parameters should have readily understood interpretations that relate to the characteristics of program, the development environment or execution environment.
- Simple to implement and program must run rapidly.

1.8 Evaluating Reliability Predictions

Need for the evaluation of competing software reliability prediction arises out of the fact that different reliability models used for the prediction produces different answers. Some techniques which help the user to decide which prediction is more trustworthy, are described [ACL86].

Even though model is an important part of the prediction system , statistical inference and prediction procedure are also vital components of the system. In fact failure of the prediction may be due to the failure of any of the above three stages. Even though in principle it is possible to study three stages separately in order to pinpoint the fault in the prediction system, because of the following reasons it is not practicable.

- Complicated nature of the model makes it difficult to apply traditional goodness of fit test, mainly due to the presence of unknown parameters.
- The underlying assumptions of a model may not always make it superior.
- Small sample properties of the estimates of the parameters are hard to obtain. Sometimes the usual asymptotic maximum likelihood(ML) estimation cannot be applied because of the finiteness of number of observable T_js.

Some techniques which can be used by the user to obtain insight into the performance of models for particular data set are,

1. Checking for self consistency: Prediction by a model for one step and n steps ahead are to be consistent with each other. But self consistency does not ensure "correctness".
2. u-plot
3. y-plot
4. Scatter plot
5. Prequential likelihood.

1.8.1 u-plot

Consider the problem of predicting current reliability. Given data $t_1, t_2, \ldots, t_{i-1}$ estimate $F_i(t) = P(T_i <= t)$ is sought. (Reliability is $R_i(t) = 1\text{-}F_i(t)$). Let the predicted value be $F_i^p(t)$. The difficulty of determining the closeness of the $F_i^p(t)$ to the true $F_i(t)$ arises from the fact $F_i(t)$ is never known. Otherwise distance measures such as those due to Kolmogorov or Cramer-Von Miser could

have been used. Analysis of closeness should only depend on the realization of T_i i.e. t_i which is a sample of size one from true distribution F_i(t).

Consider the sequence of transformations

$$u_i = F_i^p(t_i) \tag{1.44}$$

Each is a probability integral transform of the observed t_i using previously calculated predictor F_i^p based on $t_1, t_2, \ldots, t_{i-1}$. If F_i^p were to identical to the true F_i , it is easy to see that the u_is would be the realization of the independent uniform U(0,1) random variables. So the problem reduces to the question whether the sequence { u_i } "looks like" a random sample from U(0,1).

To draw a u-plot each of the given n u_is with a value between 0 and 1, is placed on the horizontal axis. The step function increases by $\frac{1}{n+1}$ at each of these points.

To examine whether u_is are uniformly distributed the c.d.f of U(0,1) (which is nothing but a line of unit slope through origin). The measure of distance used is Kolmogorov distance. If the c.d.f of u_i lies above the line of unit slope it indicates the presence of too many small u_is which in turn indicates too optimistic prediction. Similarly if the c.d.f. lies below the line of unit slope then the prediction is too pessimistic.

1.8.2 y-plot

It is observed that for a data set particular prediction system gave optimistic predictions in early stages and pessimistic predictions in later stages. These deviations are averaged out in the u-plot. So small Kolmogorov distance is observed. Thus u-plot fails to detect certain kinds of departure from reality.

Consider the transformations

$$x_i = -ln(1 - u_i) \tag{1.45}$$

{ x_i is the realization of i.i.d unit exponential random variables is u_is are realizations of U(1,0) }

That is { x_i } should looklike realization of homogeneous Poisson process. Departure will show itself as non constant rate. To test this normalize x_i. Let the normalized values be y_is.

$$y_i = \frac{\Sigma_1^i x_j}{\Sigma_1^n x_j} \tag{1.46}$$

Plot these normalized values y_i in the same manner as u_is. Analysis of the resultant graph is on the same line as u-plot.

1.8.3 Scatter Plot

The plot of u_i against i is called scatter plot. It acts as an aid to decide about the nature of the predictions. For example if the number of u_is above 0.5 is considerably less after a certain value of i say i_k then that model gives too optimistic predictions after i_k.

1.8.4 Prequential Likelihood Method

The mean square error can be written as

$$mse(\Theta^p) = var(\Theta^p) + bias(\Theta^p) \tag{1.47}$$

u-plot and y-plot procedures are an attempt to analyze the bias in predicted value F_i^p. But the absence of knowledge about F_i(t) makes it difficult to get

good measure of variability. To measure variability some approximate measures are adopted. They are as follows.

1. Median variability: Median variability is based on the median time to failure m_i. $\Sigma \frac{m_i - m_{i-1}}{m_{i-1}}$

2. Rate variability: Rate variability is based on rate of occurrence of failure (ROCOF) sequence r_i. $\Sigma \frac{r_i - r_{i-1}}{r_{i-1}}$.

3. Prequential likelihood: Prequential likelihood function is defined as follows. The predictive distribution $F_i^p(\mathrm{t})$ for T_i based on $t_1, t_2, \ldots, t_{i-1}$ will be assumed to have p.d.f.

$$f_i^p(t) = \frac{d}{dt} F_i^p(t) \tag{1.48}$$

For one step ahead predictions of $T_{j+1}, T_{j+2}, \ldots, T_{j+n}$ the prequential likelihood is given by

$$PL_n = \Pi_{j+1}^{j+n} f_i^p(t_i) \tag{1.49}$$

PL_n tends to be small if the sequence of predicted densities are either biased or noisy are both, if $F_i(\mathrm{t})$ is stationary. It behaves almost similarly even in the presence of small amount of nonstationarity. The comparison of two prediction systems, A and B, can be made via their prequential likelihood ratio

$$PLR_n = \frac{PL_n(A)}{PL_n(B)} \tag{1.50}$$

It has been shown that if PLR_n approaches ∞ as n approaches ∞, system B is discredited in favour of predictive system A.

1.9 Recalibration of Predictions

The past predictions of a model should be able to give feed back about its performance. This feed back can be used to improve the prediction capability of the model. S.Brocklehurst et.al. [SCL90] developed a recalibration technique which is model independent in nature.

Let F_i^p(t) is a predicted c.d.f of random variable T_i when the true distribution is F_i(t). Let F_i(t) be a function of predicted function F_i^p(t). that is

$$F_i(t) = G_i(F_i^p(t)) \tag{1.51}$$

considering the fact $\{ G_i \}$ is slowly changing , G_i is approximately stationary. Now G_i can be approximated with an estimate G_i^*. Then the new prediction becomes

$$F_i^{p*}(t) = G_i^*(F_i^p(t)) \tag{1.52}$$

Now observing that G_i is c.d.f. of $U_i = F_i^p(T_i)$ the estimate G_i^* can be obtained from u-plot. Simplest form of G_i^* will be the polygonal line got by joining steps of u-plot. The complete procedure for forming a recalibrated prediction for the next time to failure T_i is as follows.

1. Check that error in previous predictions is approximately stationary. (y-plot detects the nonstationarity. But recalibration seems to work well in the presence of nonstationarity).

2. Find u-plot for predictions made before T_i, i.e. based on $t_1, t_2, \ldots, t_{i-1}$, and join up the steps to form a polygonal line G_i^*.

3. Use the basic prediction system to make a "raw" prediction F_i^p(t).

4. Recalibrate the raw predictions using equation $F_i(t) = G_i^*(F_i^p(t))$.

This whole procedure can be repeated at each stage.

The recalibrated procedures are observed to be less biased than the raw ones. But this improvement may be at the cost of increase in some other deviation. This can be checked using prequential likelihood functions. It has been found that prequential likelihood ratio test sometimes rejects the recalibrated predictions in favour of raw ones. This may not be due to the failure of recalibrating technique. Consider the definition of Prequential likelihood function.

$$PL_n = \Pi_{j+1}^{j+n} f_i^p(t_i) = \Pi_{j+1}^{j+n} g_i^* F_i^p(t_i) \tag{1.53}$$

where g_i^* is the derivative of G_i^*.

Since G_i^* is polygonal line g_i^* is discontinuous which makes f_i^* discontinuous. This may generally cause PLR test to report badly on predictive accuracy of recalibrated model. It is found that polygonal line is replaced suitable smooth G_i^* PLR test favours recalibrated models. The simulation results have indicated the usefulness of recalibrated models in most of the cases even if G_i^* is polygonal line.

Chapter 2

A Software to Estimate Reliability

A software aid has been developed to estimate the reliability of a software. The program helps in the prediction of reliability quantities using one of the four software reliability models described in the last chapter. The program gets the user's choice of model, estimates the model specific parameters, calculates reliability quantities and their confidence intervals, calculates date of reaching a particular reliability goal, gives important plots and gives several quantities which can be used for comparisons.

2.1 Organization of the Program

Important modules and their interactions are shown in Figure 2.1. The emphasis is to separate the estimation procedure from the main modules so that it can be replaced by a better one, if required. Likelihood functions which are model dependent, are grouped in a single file so as to facilitate easy addition of new models.

The modules of the program are written in C. The module used for plotting makes use of SUN CGI routines. Except for this module, the program is

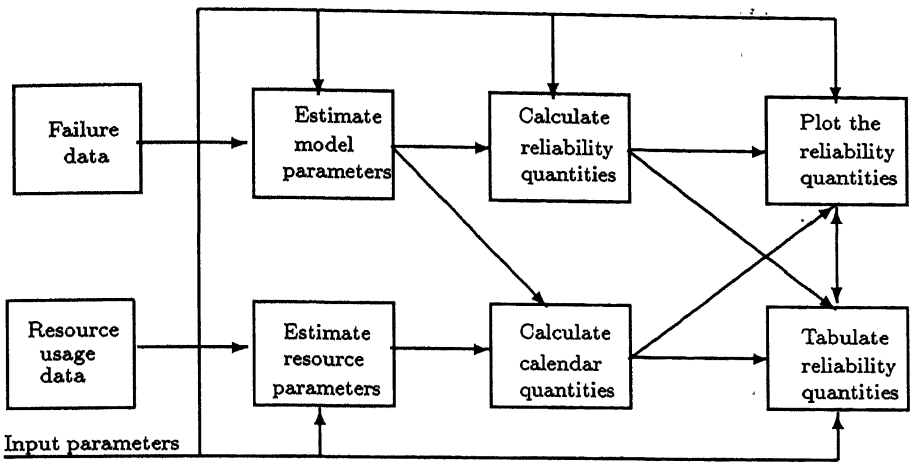


Figure 2.1: Imporatnt modules and their interactions

portable.

Files are used to transmit information between procedures. Most of these files are used for storing calculated reliability quantities. They are used as inputs for plotting appropriate graphs.

2.2 Inputs

The input to the program mainly consists of two files. One containing the failure data and the other resource usage data. The failure data can be either times between failures or times to failures. The resource usage data consists of details of amount of the three resources, namely, failure identification personnel, failure correction personnel and computer time used during a particular execution time period. It also contains data about number of faults detected and corrected in each period.

Apart from these two data, we also have several parameters which specify the model to be used, the input data form (whether times between failures or times to failures), time domain (execution or calendar), output form (tabular or plot)

and the kind of estimation of parameters (point or interval). However, most of the parameters have default values. The default values are,

- Input form: Times between failures,
- Time domain: Execution time,
- Time unit : Second, and
- Estimation: Point estimation.

Whenever required, the parameters can be changed by the user interactively, with the help of menus displayed by the program. Once the specifications are provided by the user, the program starts estimating the parameters of the model.

2.3 Estimation of the parameters

The estimation of the parameters is the most important and time consuming stage of the reliability program. It involves optimization of the function in question. If the method used for the estimation is maximum likelihood method then maximization of the function is required and, if it is least mean square error method then the minimization of the function is involved. In this program we have considered the maximum likelihood method as it has been proved to be superior to the least mean square error method, whenever large amount of data is considered.

As stated earlier this part of the program is time consuming one. Hence, any procedure used for the optimization should be very fast. Due to the nature of the object function involved, the procedure may tend to diverge, thus adding to the tardiness of the process. Hence, any procedure used for solving this maximization problem should both be fast and robust. For this purpose a

judicious combination of Newton Raphson root finding algorithm and Simplex search algorithm (suggested by [MIO87]) is used in this program.

2.3.1 Newton Raphson Method (NR)

At maximum point the unknown parameters $\vec{\beta}$ satisfy the system of equations

$$\frac{\partial lnL(\vec{\beta})}{\partial \beta_k} = 0 \qquad \text{k = 0,1,2,3 ..., n} \tag{2.1}$$

$L(\vec{\beta})$ is the likelihood function.

The NR method proposed by Carnahan and Wilkes [MIO87] provides a suitable numerical root finding procedure for solving this system of nonlinear equations. Let $U(\vec{\beta})$ be the $(w+1) \times 1$ column vector with k th element given by,

$$U_k(\vec{\beta}) = \frac{\partial lnL(\vec{\beta})}{\partial \beta_k} \tag{2.2}$$

Given starting guess $\vec{\beta_t}$ the vector defined above can be written as (using first order Taylor series expansion),

$$U(\vec{\beta}) \approx U(\vec{\beta_t}) + H(\vec{\beta_t})(\vec{\beta} - \vec{\beta_t}) \tag{2.3}$$

where $H(\vec{\beta})$ is $(w+1) \times (w+1)$ matrix with elements,

$$H_{kl}(\vec{\beta}) = \frac{\partial^2 lnL(\vec{\beta})}{\partial \beta k \partial \beta_l} \qquad \text{k,l = 0,1,2,...,n} \tag{2.4}$$

setting $U(\vec{\beta}) = 0$ and solving (2.3) we have,

$$\vec{\beta} \approx \vec{\beta_t} - H^{-1}(\vec{\beta_t}).U(\vec{\beta_t}) \tag{2.5}$$

H^{-1} is the inverse of H. Now $\vec{\beta}$ is made the new guess and the above procedure is repeated until it converges.

This method has two attractive features. Firstly, whenever it converges it converges very quickly. Secondly, if the initial guess is "near enough" to the actual root, the method surely converges. However, if the initial guess is not close enough then the method may diverge rapidly. To overcome this problem, we have to invoke another optimization method which is not so sensitive to the input guess. But the methods satisfying the above requirements are usually very slow. One such method is simplex search technique developed by Nelder and Mead [MIO87]. It has many desirable qualities which are absent in NR method. They are,

1. It is conceptually simple therefore easily tailored to meet special needs,
2. Easy to program, and
3. Robust and always converges to at least local minimum.

But it has a disadvantage that it lacks any form of acceleration. Hence simplex method is used just to refine the initial guess. This refined guess is supplied as an initial guess to the NR method. If NR method begins to diverge then the values of the parameters are restored to the value they had when they entered NR method and returned to simplex method. Simplex continues its refinement.

After the convergence criteria is satisfied, the estimated parameters are supplied to the procedures which calculate the reliability quantities. The important reliability quantities being calculated depend on the model being selected. Some of the reliability quantities are Mean Time To Failure (MTTF), median time to failure, number of errors in the system at the beginning of the testing, number of errors remaining, the failure rate, reliability etc.

Confidence intervals The confidence interval for the parameters of the model is calculated assuming normality of the joint distribution of the random variable in question. Hence, the second partial derivative of the the log likelihood function is first calculated. The bounds of confidence interval for parameter β_k are,

$$\beta_k \pm \frac{k_{1-\frac{\alpha}{2}}}{\sqrt{I_{kk}(\vec{\beta})}} \tag{2.6}$$

where I is the Fischer matrix whose elements are given by,

$$\mathbf{I_{kl}}(\vec{\beta}) = \mathbf{E}[-\frac{\partial^2 lnL(\vec{\beta}; Y_D)}{\partial\beta_k \partial\beta_l}] \tag{2.7}$$

At present 95% confidence level is being calculated.

2.3.2 Calendar component calculation

It has been proved that the execution time domain is the best domain for reliability models. Hence the input data should be provided in this domain. However, when it comes to prediction, the calendar time makes more sense. For this purpose Musa's resource usage model has been used . At present this model is available along with Musa-Okumoto logarithmic model only. It has the capability to predicted number of days required to meet the given failure intensity goal.

2.3.3 Output forms

Output provided by the program is of two kinds. The tabular form which provides the snap shot of the system, and plots of the reliability quantities which give indication of the reliability growth. The plots of MTTF and hazard rate are the important plots provided by the program. Apart from these reliability quantity plots , u-plot, y-plot and scatter plot which provide a good way of

comparing any two models, are also provided. The user can use the model which suits him most, by looking at these graphs.

The tabular form of output contains information which is useful for making comparisons among the models. Prequential likelihood value, Kolmogorov-Smirov distances of u-plot and y-plot and a measure of noise based on the ROCOF sequence have been provided. The estimated values of the parameters of the model are also given to help an interested user. The reliability quantities are failure rate, expected number of failures by a time t, an estimation of total number of faults in the program before the beginning of the testing and mean time to failure. All these quantities are not defined by all the models. Hence, the reliability quantities calculated differs from model to model. The calendar component calculation has been provided for Musa-Okumoto logarithmic model only. The goal of the software development process is to be defined in terms of final failure rate. The additional failures, additional execution time and additional calendar time required to reach the goal are calculated.

2.4 Working

The program currently runs in two modes namely, interactive mode and comparison mode. The comparison mode differs from the interactive mode mainly in one respect. Once the input parameters are set, comparison mode executes the program for the models specified in the command, keeping the parameters fixed. In the interactive mode at the end of execution of each model, user is allowed to choose a new model for which parameters can be set afresh. The program can be executed using 'rel' command. It has option -i for interactive mode and -c for comparison mode. If the -c option is used it takes an argument -a or -[numbers]. If -a option is used then all the models are executed. -[numbers] option executes the models represented by the digits of the number given. The

Jelinski-Moranda [JM] model, Littlewood-Verall [LV], Goel-Okumoto [GO] and Musa-Okumoto [MO] models have numbers 1, 2, 3 and 4 respectively.

Example:

rel -c -13 executes JM and GO models.

rel -c -124 executes JM, LV and MO models.

2.5 Shortcomings

The program has several drawbacks. Main drawback is the non-convergence of the estimation procedure. The non-convergence may be due to the insufficiency of maximum number of iterations allowed. In that case a warning is issued and whatever value is returned by previous converged estimation is taken for further calculations. Use of Fischer's information matrix is not correct under all circumstances. Under certain circumstances we have to use chi-square method for better approximations.

2.6 Conclusions and Extensions

In spite of these drawbacks the program works fairly well. The program has been tested using Musa's data for the systems T1, T2, T3, Military data [Musa] and for a data set obtained from simulation. Except at few points the program has worked satisfactorily.

For estimation procedure we can use better search techniques to speed up the process. Any interesting model which uses similar estimation procedure can be included in the program.

Chapter 3

A Reliability Model Based On Coverage Data

3.1 Motivation

Commercial concerns depend on the tools other than software reliability models for assessing the quality of the software. These tools usually consist of softwares which give information about various types of coverages of the software under testing. Percentage of branches covered, percentage of statements covered etc. are some of the important coverage measures employed. As the coverage increases more number of faults are exposed and subsequently corrected. Hence reliability of the software improves with the coverage. Looking into the reliability aspect of the software through coverage data has an advantage. It takes into account both the structure of the software and the operational profile. Number of times a piece of code is executed, reflects the operational profile. This number is provided by most of the coverage tools.

The traditional reliability models such as those discussed in chapters 1 and 2 do not take structure of the software into consideration. They follow black box approach. These models consider the reliability of the software as a function of stochastic properties of the software. If the failure data do not support their

assumptions about the stochastic properties, the models fail. Making software reliability as a function of both the deterministic properties of the structure and stochastic properties of the component failure behaviour may give, better results across different softwares.

In this chapter an attempt is made to model software growth process using coverage data. Coverage data considered is the number of times a particular piece of code is executed. While traditional software models try to capture the nature of the reliability growth through failure history, we use coverage data.

3.2 Proposed Model

Consider a directed graph where node i represents a program module or a linear code portion. A directed branch (i,j) represents a possible transfer of control from i to j. Every directed branch carries a weight p_{ij} which gives the branching probability to j when control is in i. Let $R_i(t)$ be the reliability of node i, as a function of system execution time t.

Let number of nodes in the graph be m. Without loss of generality we can assume graph has single entry and single exit node. The entry node is numbered as 1 and the exit node as m. Let n_i be the number of times node i executed. The vector N = $\{n_i\}$ forms the coverage data. In order to represent reliability as a function of coverage following assumptions are made.

3.2.1 Assumptions

1. Fault in each node manifests itself randomly according to the exponential distribution.

2. Failure rate of a fault in node i decreases with the increase in n_i.

3. The manifestation of faults in different nodes is independent of each other.

4. Faults are completely fixed as soon as they are detected.

5. Testing is random and represents operational profile.

The suitability of the assumptions to the general cases is debatable. Assumption of independence of failures may hold when nodes are modules or large code portions. However, when they are small code portions, the failures may not be independent. As our aim is to demonstrate the new idea, for the time being, exponential distribution has been assumed, even though any other distribution can be used. Assumption (2) suggests that more number of times a node is executed more defects (if any) are exposed. In actual practice faults detected may not be fixed as soon as they are detected. It has been found that the assumption of immediate fault fixing is fairely good.

3.2.2 Reliability quantities

From assumption (1) it follows that reliability of the node i is given by $R_i = e^{-\lambda_i t}$ where t is the time spent in the system. Hence λ_i gives average failure rate of the module i with respect to time spent in the system.

From assumption (2) $\lambda_i = f(n_i)$ where f is a non-increasing function of n_i. We have considered $\lambda_i = k_i e^{-\theta_i n_i}$. Here k_i can be considered as the initial failure rate of the module i with respect to the time spent in the system.

Given the reliability of the nodes, reliability of the system, R is determined through simulation. The method followed is to traverse the control flow graph of the software from node 1 to node m through different paths. Reliability along a path is taken as the product of the reliabilities of the components in that path. The reliability of the system is obtained by repeating the traversal along various paths according to the operational profile (represented by the branching probabilities) and taking the average. This simulation is repeated for the values

of t from 0 to tmax to get the graph of system reliability.

3.3 An Example

Consider the control flow graph given in Figure 3.3. The reliability functions of the modules are given with the nodes and the transitional probabilities are shown on the edges. Reliability of the software at time t = 100 is calculated. The values are taken for k_is and are shown in the graph. All θ_i s are taken as 0.09. Let, the vector N be,

$$n_1 = 200 \quad n_2 = 187 \quad n_3 = 182$$
$$n_4 = 87 \quad n_5 = 272 \quad n_6 = 52$$
$$n_7 = 133 \quad n_8 = 171 \quad n_9 = 87$$
$$n_{10} = 200$$

The reliability of the graph for t = 100 , calculated using the above described method turns out to be 0.8342. Upon further testing let N become,

$$n_1 = 400 \quad n_2 = 374 \quad n_3 = 364$$
$$n_4 = 174 \quad n_5 = 544 \quad n_6 = 104$$
$$n_7 = 266 \quad n_8 = 342 \quad n_9 = 174$$
$$n_{10} = 400$$

The reliability now increases to 0.9978

3.4 Estimation of the Parameters

The parameter to be estimated is the vector $\mathbf{K} = k_i$. k_i can be interpreted as the initial failure rate of the node i. The estimation of k_i involves two steps. First

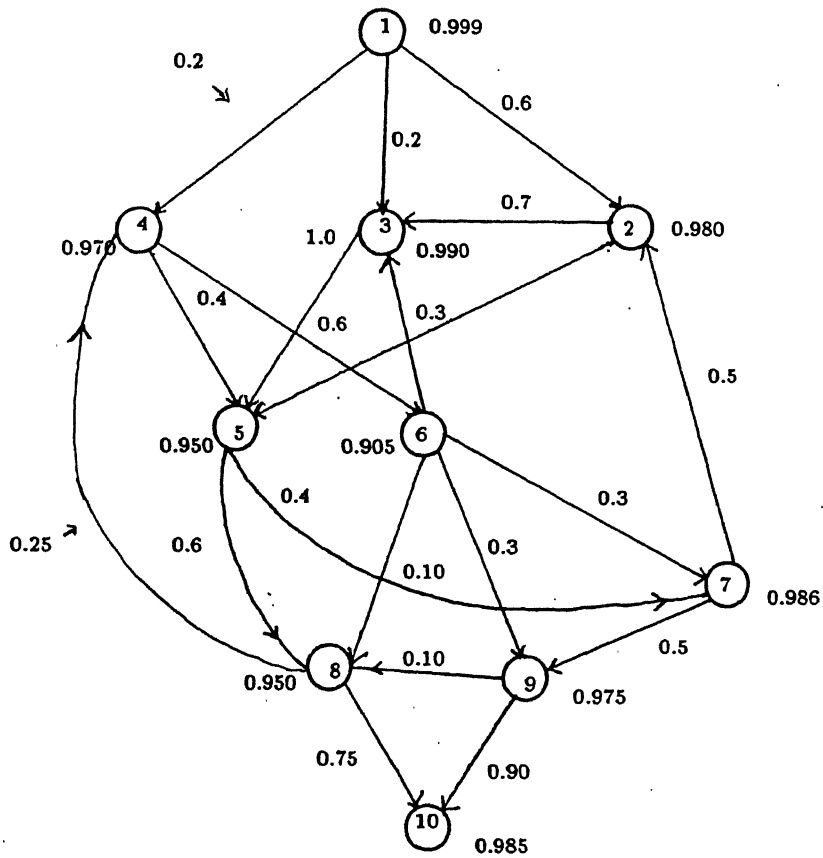


Figure 3.1: The control flow graph

use any traditional reliability model to get the failure rate λ_i' with respect to time spent in the module. Then k_i can be estimated by scaling λ_i' appropriately. For example let $\lambda_i' = 0.0001$/hr. If the module i is occupied only 20% of the system time then k_i becomes 0.00002/hr. The percentage of time, the module i occupies can be calculated either by simulation or, by using N and average time τ_i spent in each module i. That is,

$$\text{Percentage of time spent in i} = \frac{n_i \tau_i}{\Sigma n_i \tau_i} \tag{3.1}$$

The branching probabilities are calculated using data about number of inputs which have caused node i to execute, for all i. Note that N gives how many times each node is executed, which may not be same as how many inputs that caused each node to get executed. θ_i represents the rate of reduction in normalized failure intensity per run for module i, which reflects the efficiency of testing.

We hope that the difficulty in estimating the parameters can be overcome by collecting a large amount of relevent data from different projects. The available statistics about the number of faults per thousand lines of code, efficiency of testing and debugging teams can be used to calculated k_is and θ_is.

Chapter 4

Conclusions

In this thesis the software reliability and modelling has been discussed. The general techniques, various assumptions, limitations of those assumptions, general issues involved in modelling etc. are presented. Important comparison criteria are given.

We have discussed four models in detail in Chapter 1. These models require input about failure data, through which they capture failure behaviour of the software. The estimation of the parameters and reliability quantities of the models require complex calculations. To make the task of applying models easier, a software has been developed for reliability estimations. The models are popular and represent the two well known categories of the software reliability models, namely, times between failure models and failure count models. Each model predicts reliability quantities such as MTTF, hazard rate, reliability etc. The Musa logarithmic model also provides facility to get calendar time predictions. The software takes times between failures as input, and produces various quantities such as MTTF, hazard rate, reliability, expected number of failures etc.

An attempt has been made to model the reliability of a software using the

coverage data obtained during testing. The structure of the model has been taken into account while calculating the reliability of the software. Our model differs from the existing models in that it uses the number of times a particular module has been executed as the main input. The component reliabilities are combined through simulation to get the system reliability.

The validation of the assumptions and calibration of the proposed model could not be done due to the lack of data. However we feel that the analysis of the data would support our model. We hope that more and more generalizations can be done about the parameters as data from large number of projects are analyzed.

Bibliography

[ACL86] A.A. Abdel-Ghaly, P.Y. Chan and B. Littlewood,*Evaluation of competing software reliability predictions*, IEEE Tr. Software Engineering, Vol. SE-12, No. 9, pp 950-968, 1986.

[AGL85] A.L. Goel, *Software reliability modeling - Assumptions and limitations*, IEEE Tr. Software Engineering, Vol. SE-11, No. 12, pp 1411-1423, 1985.

[BLW88] B. Littlewood, *Forecasting Software Reliability*, Lecture Notes in Computer Science, 341, Sergio Bittanti(Ed.), Springer-Verlag, 1988.

[GAO79] A.L. Goel and K. Okumoto, *Time-dependent error-detection rate model for software reliability and other performance measures*, IEEE Tr. Reliability, Vol. R-28, No. 3, pp 206-211, 1979.

[ILM84] A. Iannio, B. Littlewood, J.D. Musa, K. Okumoto, *Criteria for Software Reliability Model Comparisons*, IEEE Tr. Software Engineering, Vol. SE-10, No. 6, pp 687-691, 1984.

[JAM72] Z. Jelinski and P.B. Moranda, *Software Reliability Research*, Statistical Computer performance Evaluation, W. Freibeger (Ed.), New York, Academic, 1972.

[JDM75] J.D. Musa, *A Theory Of Software Reliability And Its Application*, IEEE Tr. Software Engineering, Vol. SE-1, No. 3, pp 312-327, 1975.

[JDM79] J.D. Musa, *Software Reliability Data,* report, Data and Analysis Center for Software, Rome Air Development Center, Rome, NY, 1979.

[JDM85] J.D. Musa, *Software reliability*, Hand Book of Software Engineering, Ramamoorthy(Ed.), 1985.

[KLM83] P.A. Keiller, B. Littlewood, D.R. Miller and A. Sofer, *On The Quality of Software Reliability Prediction,* (J.K. Skwirzynski, Editor), Electronic Systems Effectiveness and Life Cycle Costing, NATO ASI Series, F3, Springer-Verlag, Heidelberg, pp 441-460, 1983.

[LAV73] B. Littlewood and J.L. Verrall, *A Bayesian Reliability Growth Model for Computer Software*, J. Royal Statist. Soc., C(Applied Statistics), Vol. 2, pp 332-346, 1973.

[MAA92] M.R. Lyu and A. Nikora, *Applying Reliability Models More Effectively*, IEEE Software, No. 7, pp 43-52, 1992.

[MAO84] J.D. Musa and K. Okumoto, *A Logarithmic Execution time model for software reliability measurement*, Proc. 7th International conference on Software Engineering, Orlando, Florida, March 26-29, pp 230-238, 1984.

[MAO84a] J.D. Musa and K. Okumoto, *A Comparison of Time Domains for Software Reliability Measurement,"* Journal of Systems and Software, Vol. 4, No. 4, pp 230-238, 1984.

[MAO88] J.D. Musa and K. Okumoto, *Application Of Basic and Logarithmic Poisson Execution Time Models in Software Reliability Measurement*, Lecture Notes in Computer Science, 341, Sergio Bittanti(Ed.), Springer-Verlag, 1988.

[MIO87] J.D. Musa, A. Iannino and K. Okumoto, *Software Reliability, Measurement, Prediction, Application*, McGraw-Hill, 1987.

[MOR75] Z. Jelinski and P.B. Moranda, *Prediction of Software Reliability During Debugging*, Proceedings of Annual Reliability and Maintainability Symposium, Washington D.C, pp 327-332, 1975.

[SAW73] G.J. Schick and R.W. Wolverton, *Assesment of Software Reliability,* Proceedings of Operation Research, Physica Verlag, Wurzburg-Wein, pp 395-422, 1973

[SCL90] Sarah Brocklehurst, P.Y. Chan, B. Littlewood and J. Snell, *Recalibrating Software Reliability Models*, IEEE Tr. Software Engineering, Vol. 16, No. 4, pp 458-470, 1990.

[SEB88] Sergio Bittanti, *An Introduction to Software Reliability Modelling*, Lecture Notes in Computer Science, 341, Sergio Bittanti(Ed.), Springer-Verlag, 1988.

[SHO72] M.L. Shooman, *Probabilistic Models for Software Reliability Prediction*, Statistical Computer Performance Evaluation, Academic, New York, pp 485-502, 1972.